---

title: Rigil Testnet description: Migrating your work to the Rigil testnet keywords: - test - build - suave - create - deploy - transact

---

import RPCButton from '@site/src/components/RPCButton/index';

Rigil

All our tutorials have taken you through how to work with SUAVE locally, and the `spell` tool is intended only for local use.

So, now that we have all these cool contracts, how can we deploy them to the Rigil testnet and begin transacting there?

:::info

You can look through all the tools and examples already in use on Rigil in **our community directory**.

:::

Chain info

- Block Explorer
- Faucet
- EthStats
- Technical Specs
- chainId: 16813125
- Rigil Kettle Address: 0x03493869959c866713c33669ca118e774a30a0e5

We have RPC nodes you can connect to:

Deploy Contracts

1. Get rETH from the faucet.
2. The easiest way to deploy contracts you've already made is to go on using Forge. From the root of your contracts directory, you can run:

bash forge create --rpc-url https://rpc.rigil.suave.flashbots.net --legacy \ --private-key <your_funded_priv_key> <your_contract_name>.sol:<your_contract_name>

Note the `--legacy` flag: transactions on SUAVE are not EIP1559 compliant, which you will likely need to take into account no matter which smart contract framework you use.

You should see something like this printed to your console:

bash ...relevant compilation results... Deployer: 0xBE69d72ca5f88aCba033a063dF5DBe43a4148De0 Deployed to: 0xcbdF0322Cd79212e10b0dB72D775aE05B99c1796 Transaction hash: 0x9ae80af40bdafbc706108446dbbf7761a59f5bf544b46c97b9b0851dddaa3927

Sending Transactions

We support both a Golang SDK and a typescript SDK to make sending transactions and confidential compute requests easy.

Golang SDK

The most effective way to begin with the Golang SDK is to use the same `framework.go` used for all the `suapp-examples`, as it implements everything you could need for interacting with your contracts.

It's use is demonstrated in the `main.go` in each example, from which you should be able to learn everything from deploying contracts to sending custom confidential compute requests from various different actors.

Typescript SDK

We generally use `bun` to manage dependencies for our typescript SDK.

1. Create a file called `index.ts` in the root of your directory.
2. Copy and paste this, making the adjustments specified in the comments:

```ts import {http} from '@flashbots/suave-viem'; import {getSuaveWallet, TransactionRequestSuave, SuaveTxRequestTypes} from '@flashbots/suave-viem/chains/utils'

const SUAVE_RPC_URL = 'https://rpc.rigil.suave.flashbots.net'; // Change this to a private key with rETH you get from https://faucet.rigil.suave.flashbots.net/ const PRIVATE_KEY = '0x

const wallet = getSuaveWallet({ transport: http(SUAVE_RPC_URL), privateKey: PRIVATE_KEY, });

const ccr: TransactionRequestSuave = { confidentialInputs: '0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000fd7b22626c6f636b4e756d626572223a22
kettleAddress: '0x03493869959C866713C33669cA118E774A30A0E5', to: '0x8f21Fdd6B4f4CacD33151777A46c122797c8BF17', gasPrice: 10000000000n, gas: 420000n, type:
SuaveTxRequestTypes.ConfidentialRequest, chainId: 16813125, data:
'0x54353f2f0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000006000000000000000000000000
// Data payload for the transaction };

const res = await wallet.sendTransaction(ccr); console.log(`sent ccr! tx hash: ${res} ) `` 3. Run `bun index.ts` and check your console for tx hash of your first CCR on Rigil.

If you'd like to see how to construct the `confidentialInputs` and `data` fields within the context of a web application, you can fork `this file` as an exemplary starting point.

### Node

If you haven't used bun before, or are unfamiliar with typescript, here is a simplified JS file you can run using Node.

1. Create a file called `index.js` in the root of your directory.
2. Copy and paste this, making the adjustments specified in the comments:

```js const { http } = require('@flashbots/suave-viem'); const { getSuaveWallet } = require('@flashbots/suave-viem/chains/utils');

const SUAVE_RPC_URL = 'https://rpc.rigil.suave.flashbots.net'; // Change this to a private key with rETH you get from https://faucet.rigil.suave.flashbots.net/ const PRIVATE_KEY = '0x

const wallet = getSuaveWallet({ transport: http(SUAVE_RPC_URL), privateKey: PRIVATE_KEY, });

async function sendCCR() { const ccr = { confidentialInputs:
'0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000fd7b22626c6f636b4e756d626572223a22
kettleAddress: '0x03493869959C866713C33669cA118E774A30A0E5', to: '0x8f21Fdd6B4f4CacD33151777A46c122797c8BF17', gasPrice: 10000000000n, gas: 420000n, type: "0x43", // SUAVE
transaction request type chainId: 16813125, data:
'0x54353f2f0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000006000000000000000000000000
};

const res = await wallet.sendTransaction(ccr);
console.log(`sent ccr! tx hash: ${res}`);

}

sendCCR().catch(console.error); ```

1. Run `node index.js` and check your console for tx hash of your first CCR on Rigil.